

AD-A204 449

DTIC FILE COPY

2

AVF Control Number: AVF-VSR-013
SZT-AVF-013

Ada COMPILER
VALIDATION SUMMARY REPORT:
Certificate Number: 88032011.09057
TELESOFT / Intel Corp. / Telelogic AB
Ada386
Version 3.20
VAX 8630, Intel 386

Completion of On-Site Testing:
88-03-20

Prepared By:
IABG m.b.H., Dept SZT
Einsteinstrasse 20
8012 Ottobrunn
West Germany

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

DTIC
ELECTE
S 13 FEB 1989 D
E

Ada is a registered trademark of the United States Government
(Ada Joint Program Office).

This document has been approved
for public release and sale in
distribution is unlimited.

89 2 13 110

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	12. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report; TELESOFT/Intel Corp./Telelogic AB, Ada386, Version 3.20, VAX 8630 (Host) to Intel 386 (Target). (88032011.09057)		5. TYPE OF REPORT & PERIOD COVERED 20 March 1988 to 20 March 1988
7. AUTHOR(s) IABG, Ottobrunn, Federal Republic of Germany.		6. PERFORMING ORG. REPORT NUMBER
8. PERFORMING ORGANIZATION AND ADDRESS IABG, Ottobrunn, Federal Republic of Germany.		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) IABG, Ottobrunn, Federal Republic of Germany.		12. REPORT DATE 20 March 1988
		13. NUMBER OF PAGES 40 p.
		15. SECURITY CLASS (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report) UNCLASSIFIED		
18. SUPPLEMENTARY NOTES		
19. KEYWORDS (Continue on reverse side if necessary and identify by block number) Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD- 1815A, Ada Joint Program Office, AJPO		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Ada 386, Version 3.20, TELESOFT/Intel Corp./Telelogic AB, IABG, VAX 8630 under VMS, Version 4.6 (Host) to Intel 386-116 CPU Board (bare machine) (Target), ACVC 1.9.		

DD FORM 1473

EDITION OF 1 NOV 65 IS OBSOLETE

1 JAN 78

S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Ada Compiler Validation Summary Report:

Compiler Name: Ada386
Compiler Version: 3.20

Certificate Number: 88032011.09057

Host: VAX 8630 under VMS, Version 4.6

Target: Intel 386-116 CPU Board bare machine

Testing Completed 88-03-20 Using ACVC 1.9

This report has been reviewed and is approved.

H. H. Hummel

IABG m.v.H., Dept SZT
Dr. H. Hummel
Einsteinstrasse 20
8012 Ottobrunn
West Germany



John F. Kramer

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC 20301

CONTENTS

CHAPTER 1	INTRODUCTION	
1.1	PURPOSE OF THIS VALIDATION SUMMARY REPORT	1-2
1.2	USE OF THIS VALIDATION SUMMARY REPORT	1-2
1.3	REFERENCES	1-3
1.4	DEFINITION OF TERMS	1-4
1.5	ACVC TEST CLASSES	1-5
CHAPTER 2	CONFIGURATION INFORMATION	
2.1	CONFIGURATION TESTED	2-1
2.2	IMPLEMENTATION CHARACTERISTICS	2-2
CHAPTER 3	TEST INFORMATION	
3.1	TEST RESULTS	3-1
3.2	SUMMARY OF TEST RESULTS BY CLASS	3-1
3.3	SUMMARY OF TEST RESULTS BY CHAPTER	3-2
3.4	WITHDRAWN TESTS	3-2
3.5	INAPPLICABLE TESTS	3-2
3.6	TEST, PROCESSING, AND EVALUATION MODIFICATIONS	3-5
3.7	ADDITIONAL TESTING INFORMATION	3-6
3.7.1	Prevalidation	3-6
3.7.2	Test Method	3-6
3.7.3	Test Site	3-8
APPENDIX A	DECLARATION OF CONFORMANCE	
APPENDIX B	APPENDIX F OF THE Ada STANDARD	

APPENDIX C TEST PARAMETERS

APPENDIX D WITHDRAWN TESTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

(KR) ←

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- . To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- . To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- . To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 88-03-20 at Telelogic AB at Nynashamn, Sweden.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

IABG m.b.H., Dept SZT
Einsteinstrasse 20
8012 Ottobrunn
West Germany

INTRODUCTION

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

INTRODUCTION

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the <u>Ada Compiler Validation Procedures and Guidelines</u> .
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect

because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

INTRODUCTION

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: Ada386, Version 3.20

ACVC Version: 1.9

Certificate Number: 88032011.09057

Host Computer:

Machine:	VAX 8630
Operating System:	VMS, Version 4.6
Memory Size:	20 MB

Target Computer:

Machine:	Intel 386-116 CPU Board
Operating System:	bare machine
Memory Size:	2 MB

Communications Network: RS 232 interface

CONFIGURATION INFORMATION

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

- . Capacities.

The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- . Universal integer calculations.

An implementation is allowed to reject universal integer calculations having values that exceed `SYSTEM.MAX_INT`. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- . Predefined types.

This implementation supports the additional predefined types `LONG_INTEGER` and `LONG_FLOAT` in the package `STANDARD`. (See tests B86001C and B86001D.)

- . Based literals.

An implementation is allowed to reject a based literal with a value exceeding `SYSTEM.MAX_INT` during compilation, or it may raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` during execution. This implementation raises `NUMERIC_ERROR` during execution. (See test E24101A.)

- . Expression evaluation.

Apparently some default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

Sometimes `NUMERIC_ERROR` is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Apparently `NUMERIC_ERROR` is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is gradual. (See tests C45524A..Z.)

. Rounding.

The method used for rounding to integer is apparently round to even. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round to even. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round away from zero. (See test C4A014A.)

. Array types.

An implementation is allowed to raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` for an array having a `'LENGTH` that exceeds `STANDARD.INTEGER'LAST` and/or `SYSTEM.MAX_INT`. For this implementation:

No exception is raised when an array type or subtype with more than `SYSTEM.MAX_INT` components is declared. (See test C36003A.)

No exception is raised when `'LENGTH` is applied to an array type with `INTEGER'LAST + 2` components. (See test C36202A.)

No exception is raised when `'LENGTH` is applied to an array type with `SYSTEM.MAX_INT + 2` components. (See test C36202B.)

A packed `BOOLEAN` array having a `'LENGTH` exceeding `INTEGER'LAST` raises no exception. (See test C52103X.)

A packed two-dimensional `BOOLEAN` array with more than `INTEGER'LAST` components raises `CONSTRAINT_ERROR` when the length of a dimension is calculated and exceeds `INTEGER'LAST`. (See test C52104Y.)

CONFIGURATION INFORMATION

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises no exception. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Aggregates.

In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

. Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is used by a test in a way that violates a restriction, then the implementation must reject it.

CONFIGURATION INFORMATION

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are not supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are supported. Allocated objects must have a minimum allocation size of 16 bits. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are supported. (See tests A39005E and C87B62C.)

Record representation clauses are supported. An alignment of 16 for the record is required. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are supported. (See test C87B62A.)

. Pragas.

The pragma INLINE is supported for procedures. The pragma INLINE is not supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

. Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behavior for SEQUENTIAL_IO, DIRECT_IO, and TEXT_IO.

CONFIGURATION INFORMATION

. Generics.

Generic subprogram declarations and bodies can be compiled in separate compilations. (See test CA1012A.)

Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

Generic package declarations and bodies can be compiled in separate compilations. (See tests BC3204C and BC3205D. These tests demonstrate that the compiler is able to compile generic package declarations and bodies separately, but these tests are not applicable for the reason given below and in 3.5.)

This implementation creates a dependence between a generic body and those units which instantiate it. If a generic body is missing or obsolete, and this body has been instantiated, then the instantiations become obsolete, when the actual generic body is compiled.

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tested, 27 tests had been withdrawn because of test errors. The AVF determined that 421 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation and 174 executable tests that use file operations not supported by the implementation. Modifications to the code, processing, or grading for 20 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT -----	TEST CLASS						TOTAL -----
	A	_B_	_C_	_D_	_E_	_L_	
Passed	105	1046	1449	17	12	45	2674
Inapplicable	5	5	404	0	6	1	421
Withdrawn	3	2	21	0	1	0	27
TOTAL	113	1053	1874	17	19	46	3122

TEST INFORMATION

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER														TOTAL
	__2__	__3__	__4__	__5__	__6__	__7__	__8__	__9__	__10__	__11__	__12__	__13__	__14__	_____	
Passed	190	494	537	245	166	98	141	327	132	36	232	3	73	2674	
Inapplicable	14	78	137	3	0	0	2	0	5	0	2	0	180	421	
Withdrawn	2	14	3	0	0	1	2	0	0	0	2	1	2	27	
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122	

3.4 WITHDRAWN TESTS

The following 27 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	E28005C	C34004A	C35502P	A35902C
C35904A	C35904B	C35A03E	C35A03R	C37213H
C37213J	C37215C	C37215E	C37215G	C37215H
C38102C	C41402A	C45332A	C45614C	A74106C
C85018B	C87B04B	CC1311B	BC3105A	AD1A01A
CE2401H	CE3208A			

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 421 tests were inapplicable for the reasons indicated:

- C35508I..J (2 tests) and C35508M..N (2 tests) use enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1). These clauses are not supported by this compiler.

TEST INFORMATION

- . C35702A uses SHORT_FLOAT which is not supported by this implementation.
- . A39005B uses length clauses with SIZE specifications for enumeration types which are not supported by this compiler.
- . A39005G uses a record representation clause which is not supported by this compiler.
- . The following tests use SHORT_INTEGER, which is not supported by this compiler:

C45231B	C45304B	C45502B	C45503B	C45504B
C45504E	C45611B	C45613B	C45614B	C45631B
C45632B	B52004E	C55B07B	B55B09D	

- . C45231D requires a macro substitution for any predefined numeric types other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT, and LONG_FLOAT. This compiler does not support any such types.
- . C45531M, C45531N, C45532M, and C45532N use fine 48-bit fixed-point base types which are not supported by this compiler.
- . C45531O, C45531P, C45532O, and C45532P use coarse 48-bit fixed-point base types which are not supported by this compiler.
- . C4A012B contains a variable, which is never used (dead variable). The compiler legitimately does not generate code for operations with this variable. As a result, at execution time neither of the exceptions this program tests for are raised, and the test fails.
- . B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.
- . C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.
- . CA2009C, CA2009F, BC3204C, and BC3205D contain instantiations of generics in cases where the body is not available at the time of the instantiation. As allowed by AI-00408/11, this compiler creates a dependency on the missing body so that when the actual body is compiled, the unit containing the instantiation becomes obsolete.
- . CA3004F, EA3004D, and LA3004B use the INLINE pragma for functions, which is not supported by this compiler.

TEST INFORMATION

- AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.
- AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.
- The following 174 tests are inapplicable because sequential, text, and direct access files are not supported.

CE2102C	CE2102G..H(2)	CE2102K	CE2104A..D(4)
CE2105A..B(2)	CE2106A..B(2)	CE2107A..I(9)	CE2108A..D(4)
CE2109A..C(3)	CE2110A..C(3)	CE2111A..E(5)	CE2111G..H(2)
CE2115A..B(2)	CE2201A..C(3)	CE2201F..G(2)	CE2204A..B(2)
CE2208B	CE2210A	CE2401A..C(3)	CE2401E..F(2)
CE2404A	CE2405B	CE2406A	CE2407A
CE2408A	CE2409A	CE2410A	CE2411A
AE3101A	CE3102B	EE3102C	CE3103A
CE3104A	CE3107A	CE3108A..B(2)	CE3109A
CE3110A	CE3111A..E(5)	CE3112A..B(2)	CE3114A..B(2)
CE3115A	CE3203A	CE3301A..C(3)	CE3302A
CE3305A	CE3402A..D(4)	CE3403A..C(3)	CE3403E..F(2)
CE3404A..C(3)	CE3405A..D(4)	CE3406A..D(4)	CE3407A..C(3)
CE3408A..C(3)	CE3409A	CE3409C..F(4)	CE3410A
CE3410C..F(4)	CE3411A	CE3412A	CE3413A
CE3413C	CE3602A..D(4)	CE3603A	CE3604A
CE3605A..E(5)	CE3606A..B(2)	CE3704A..B(2)	CE3704D..F(3)
CE3704M..O(3)	CE3706D	CE3706F	CE3804A..E(5)
CE3804G	CE3804I	CE3804K	CE3804M
CE3805A..B(2)	CE3806A	CE3806D..E(2)	CE3905A..C(3)
CE3905L	CE3906A..C(3)	CE3906E..F(2)	

Results of running a subset of these tests showed that the proper exceptions are raised for unsupported file operations.

- The following 201 tests require a floating-point accuracy that exceeds the maximum of 15 digits supported by this implementation:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 16 Class B tests, 3 Class C tests, and 1 Class E test.

The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B27005A	B71001E	B71001K	B71001Q	B71001W
B97101A	B97101E	BA1101C	BA3006A	BA3006B
BA3007B	BA3008A	BA3008B	BA3013A	

For the following class B and E tests a "PRAGMA LIST(ON);" was inserted at the beginning of the program in order to have a full compilation listing produced. If a "PRAGMA LIST(ON)" is given anywhere in a program source the compiler assumes that the compilation listing is to be suppressed until the "PRAGMA LIST(ON)" appears in the program source, even if the compilation was started with full listing option. The AVO regards this interpretation of the Ada Standard as unique and announces further discussions about the interpretation of PRAGMA LIST.

B28001R	B28001V	E28002D
---------	---------	---------

The following class C tests were modified for the reasons indicated:

- C45651A requires that the result of the expression in line 227 be in the range given in line 228; however this range excludes some acceptable results. This implementation passes all other checks of this test, and the AVO ruled that this test is passed.
- C46014A (like C4A012B) contains a variable, that is never used in the program. To demonstrate an acceptable behavior of the test a line "86.5" was inserted into the source of C46014A:
"IF IDENT_INT (I1) = 0 THEN COMMENT ("I1 = 0"); END IF;".
With this modification the test passes.
- C96001A assumes that DURATION'SMALL >= SYSTEM'TICK; however, the Ada standard does not require such a relation. This implementation executes delay statements with greater accuracy than CALENDAR'CLOCK can resolve, and so the check on line 97 is failed. This implementation passes all other checks of this test,

TEST INFORMATION

and the AVO ruled that the test is passed.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the Ada386 was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the Ada386 using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX 8630 host operating under VMS, Version 4.6, and a Intel 386-116 CPU Board target (bare machine). The host and target computers were linked via RS 232 interface.

A magnetic tape containing all tests except for the withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were not included in their modified form on the magnetic tape.

The contents of the magnetic tape were loaded directly onto the host computer. After the test files were loaded to disk, the full set of tests was compiled and linked on the VAX 8630, and all executable tests were run on the Intel 386-116 CPU Board. Object files were linked on the host computer, and executable images were transferred to the target computer via RS 232 interface. Results were printed from the host computer, with results being transferred to the host computer via RS 232 interface.

The compiler was tested using command scripts provided by Telelogic AB and reviewed by the validation team. The compiler was tested using the following option settings:

TEST INFORMATION

Option	Effect
MONITOR	verbose
PROCEED	continue compilation despite errors, no continuation prompting at each error
LIST	source code listing (only class B tests)
VIRTUAL_SPACE=3000	set virtual space of library manager greater than default
LIBFILE="BIN"	each bin gets its own library list allowing multiple compiles in same directory and allowing new library for each compilation, enhancing testing speed
OPT="<options file>"	options file appropriate for target configuration (only executable tests)
LOAD_MODULE	output executable in Tetesoft execute from suitable for downloading to enhance speed (only executable tests)
OPTIMIZE	equivalent to "OPTIMIZE=ALL", in which ALL stands for "PARALLEL, RECURSE, INLINE, AUTOINLINE"
PARALLEL	indicates that one or more of the subprograms being optimized may be called from parallel tasks
RECURSE	indicates that one or more of the subprograms interior to the unit/collection being optimized could be called recursively by an exterior subprogram
INLINE	enables inline expansion of those subprograms marked with an inline pragma or generated by the compiler
AUTOINLINE	enables automatic inline expansion of any subprogram called from only one place, as well as those marked by an inline pragma or generated by the compiler

Tests were compiled, linked, and executed (as appropriate) using a single host computer and a single target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

TEST INFORMATION

3.7.3 Test Site

Testing was conducted at Telelogic AB at Nynashamn, Sweden and was completed on 88-03-20.

APPENDIX A
DECLARATION OF CONFORMANCE

**Telelogic AB has submitted the following Declaration of
Conformance concerning the Ada386.**

DECLARATION OF CONFORMANCE

Compiler Implementer: TELESOFT
Ada Validation Facility: IABG, Munich, West-Germany
Ada Compiler Validation Capability (ACVC) Version: 1.9

Base Configuration

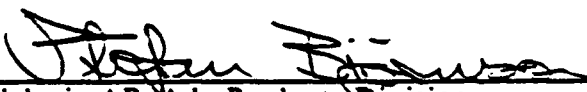
Base Compiler Name:	Ada386
Version:	3.20
Host Architecture ISA:	VAX 8530
OS & VER #:	VMS 4.6
Target Architecture ISA:	Intel 80386, implemented on Intel 386-100 board
OS & VER #:	Bare Machine

Derived Compiler Registration

Derived Compiler Name:	Ada386
Version:	3.20
Host Architecture ISA:	MicroVAX II, MicroVAX 2000, MicroVAX 3500, MicroVAX 3600, MicroVAX I, VAXStation I, VAXStation II, VAXStation 2000, VAXStation 3200, VAXStation 3500, VAXServer 3500, VAXServer 3600, VAXServer 3602, VAX-11/725, VAX-11/730, VAX-11/750, VAX-11/780, VAX-11/782, VAX-11/785, VAX 8200, VAX 8300, VAX 8500, VAX 8600, VAX 8530, VAX 8650, VAX 8700, VAX 8800, VAX 8974, VAX 8978
OS & VER #:	VMS 4.5 & VMS 4.6
Target Architecture ISA:	Intel 80386, implemented on the Intel 386-116 board, including Intel 80387 floating-point processor.
OS & VER #:	Bare Machine

IMPLEMENTER'S DECLARATION


I, the undersigned, representing TELESOFT, have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that TELESOFT and INTEL CORPORATION are the joint owners of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for the Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.


Telelogic AB, Ada Products Division
Stefan Bjornson
Manager, Systems Software

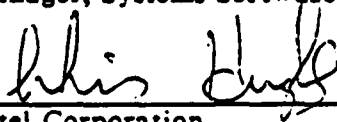
Date: 27 May 1988

OWNER'S DECLARATION

We, the undersigned, representing TELESOFT and INTEL CORPORATION as joint owners take full responsibility for implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. We further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. We declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.


Telelogic AB, Ada Products Division
Stefan Bjornson
Manager, Systems Software

Date: 27 May 1988


Intel Corporation
Chris Hughes
General Manager, DTO

Date: 4/29/88



APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the Ada386, Version 3.20, are described in the following sections, which discuss topics in Appendix F of the Ada Standard. Implementation-specific portions of the package STANDARD are also included in this appendix.

APPENDIX F OF THE LANGUAGE REFERENCE MANUAL

- 1 Implementation Dependent Pragmas
- 2 Implementation Dependent Attributes
- 3 Specification of Package SYSTEM
- 4 Restrictions on representation clauses
- 5 Implementation dependent naming
- 6 Interpretation of expressions in address clauses
- 7 Restrictions on unchecked conversions
- 8 I/O Package characteristics
- 9 Definition of STANDARD

APPENDIX F

1. Predefined Pragma

pragma LIST(ON|OFF);

It may appear anywhere a pragma is allowed. The pragma has the effect of generating the source compilation listing from the argument ON is given until the opposite argument OFF is given within the same compilation.

Implementation Dependent Pragmas

pragma COMMENT(<string_literal>);

It may only appear within a compilation unit. The pragma comment has the effect of embedding the given sequence of characters in the object code of the compilation unit.

pragma LINKNAME(<subprogram_name>, <string_literal>);

It may appear in any declaration section of a unit. This pragma must also appear directly after an interface pragma for the same <subprogram_name>. The pragma linkname has the effect of making string_literal apparent to the linker.

pragma INTERRUPT(Function_Mapping);

It may only appear immediately before a simple accept statement, a while loop directly enclosing only a single accept statement, or a select statement that includes an interrupt accept alternative. The pragma interrupt has the effect that entry calls to the associated entry, on behalf of an interrupt, are made with a reduced call overhead.

pragma IMAGES (<enumeration_type>, DEFERRED|IMMEDIATE);

The pragma and the enumeration type declaration for which the pragma applies must both occur immediately within the same declarative part, package specification, or task specification; the enumeration type declaration must occur before the clause. If a user tries to associate two IMAGES pragmas with the same enumeration type, the second pragma will be ignored and a warning will be issued. The enumeration type must not be a derived type or a subtype. PRAGMA IMAGES has the effect that enumeration literals are either generated directly at point of declaration (IMMEDIATE) or later, at point of reference(s) (DEFERRED) for the given type.

2. Implementation Dependent Attributes

There are no implementation dependent attributes.

3. Specification of Package SYSTEM

PACKAGE System IS

TYPE Address is Access Integer;

TYPE Subprogram_Value is PRIVATE;

TYPE Name IS (TeleGen2);

System_Name : CONSTANT name := TeleGen2 ;

APPENDIX F, Cont.

Storage_Unit : CONSTANT := 8;
 Memory_Size : CONSTANT := (2 ** 31) - 1;

-- System-Dependent Named Numbers:

Min_Int : CONSTANT := -(2 ** 31);
 Max_Int : CONSTANT := (2 ** 31) - 1;
 Max_Digits : CONSTANT := 15;
 Max_Mantissa : CONSTANT := 31;
 Fine_Delta : CONSTANT := 1.0 / (2 ** Max_Mantissa);
 Tick : CONSTANT := 10.0E-3;

-- Other System-Dependent Declarations

SUBTYPE Priority IS Integer RANGE 0 .. 63;

Max_Text Io_Count : CONSTANT := Max_Int;
 Max_Text Io_Field : CONSTANT := 1000;

PRIVATE

TYPE Subprogram_Value IS
 RECORD
 Proc_addr : Address;
 Static_link : Address;
 END RECORD;

END System;

4. Restrictions on Representation Clauses

The Compiler supports the following representation clauses:

Length Clauses: for enumeration and derived integer types 'SIZE
 attribute (LRM 13.2(a))

Length clauses: for access types 'STORAGE_SIZE attribute (LRM 13.2(b))

Length Clauses: for task types 'STORAGE_SIZE attribute (LRM 13.2(c))

Length clauses: for fixed point types 'SMALL attribute (LRM 13.2(d))

Enumeration clauses: for character and enumeration types other than
 character and boolean (LRM 13.3)

Record representation clauses (LRM 13.4)

Address Clauses: for objects and entries (LRM 13.5(a)(c))

APPENDIX F, Cont.

This compiler does NOT support the following representation clauses:

Enumeration clauses: for boolean (LRM 13.3)

Address clauses for subprograms, packages, and tasks (LRM 13.5(b))

Note: The VAX/E386 compiler contains a restriction that allocated objects must have a minimum allocation size of 16 bits.

5. Implementation dependent naming conventions

There are no implementation-generated names denoting implementation dependent components.

6. Expressions that appear in address specifications are interpreted as the first storage unit of the object.

7. Restrictions on Unchecked Conversions

Unchecked conversions are allowed between any types unless the target type is an unconstrained record or array type.

8. I/O Package Characteristics

Instantiations of **DIRECT_IO** and **SEQUENTIAL_IO** are supported with the following exceptions:

- * Unconstrained array types.

- * Unconstrained types with discriminants without default values.

- * In **DIRECT_IO** the type **COUNT** is defined as follow:

type **COUNT** is range 0..2_147_483_647;

- * In **TEXT_IO** the type **COUNT** is defined as follows:

type **COUNT** is range 0..2_147_483_646;

APPENDIX F, Cont.

* In TEXT_IO the subtype FIELD is defined as follows:

subtype FIELD is INTEGER range 0..1000;

9. Definition of STANDARD

For this target system the numeric types and their properties are as follows:

Integer types:

INTEGER

size = 16

first = -32768

last = +32767

LONG_INTEGER

size = 32

first = -2147483648

last = +2147483647

Floating-point types:

FLOAT

size = 32

digits = 6

'small = 2.58494E-26

'large = 1.93428E+25

APPENDIX F, Cont.

```
machine_radix = 2
machine_mantissa = 24
machine_emin = -125
machine_emax = +127
```

LONG_FLOAT

```
size = 64
digits = 15
'small = 1.94469227433161E-62
'large = 2.57110087081438E+61
machine_radix = 2
machine_mantissa = 53
machine_emin = -1021
machine_emax = +1023
```

Fixed-point types:

SHORT_FIXED

```
size = 16
delta = 2#1.0#e-15
first = -1.00000
last = +1.0 - 2#1.0#e-15
```

FIXED

```
size = 32
delta = 2#1.0#e-31
first = -1.00000
last = +1.0 - 2#1.0#e-31
```

DURATION

```
size = 32
delta = 2#1.0#e-14
first = -86400
last = +86400
```

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning_____	Value_____
\$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..199 => 'A', 200 => '1')
\$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..199 => 'A', 200 => '2')
\$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..100=>'A',101=>'3',102..200=>'A')
\$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..100=>'A',101=>'4',102..200=>'A')
\$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..197=>'0') & "298"

TEST PARAMETERS

Name and Meaning	Value
#BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..195=>'0') & "690.0"
#BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1 .	(1=>'',2..101=>'A',102=>'')
#BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1 .	(1=>'',2..100=>'A',101=>'1',102=>'')
#BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	(1..180 => ' ')
#COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST .	2147483646
#FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST .	1000
#FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	X)JZ!@#%^&*~Y
#FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	XYZ*
#GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION .	86401.0

TEST PARAMETERS

<u>Name_and_Description</u>	<u>Value</u>
#GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	131072.0
#ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	BAD_CHARACTER+^/Z
#ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	(1..120 => 'A')
#INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-32768
#INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	32767
#INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST + 1.	32768
#LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-86401.0
#LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-131072.0
#MAX_DIGITS Maximum digits supported for floating-point types.	15
#MAX_IN_LEN Maximum input line length permitted by the implementation.	200
#MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
#MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648

TEST PARAMETERS

Name and Meaning	Value
\$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	"2:" & (3..197=>'0') & "11:"
\$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	"16:" & (4..196=>'0') & "F.E:"
\$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	(1=>'"',2..199=>'A',200 =>'")
\$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483647
\$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.	\$NAME
\$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFFF#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 27 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- B2B003A: A basic declaration (line 36) wrongly follows a later declaration.
- E2B005C: This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ARG.
- C34004A: The expression in line 168 wrongly yields a value outside of the range of the target type T, raising CONSTRAINT_ERROR.
- C35502P: Equality operators in lines 62 & 69 should be inequality operators.
- A35902C: Line 17's assignment of the nominal upper bound of a fixed-point type to an object of that type raises CONSTRAINT_ERROR, for that value lies outside of the actual range of the type.
- C35904A: The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.
- C35904B: The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may in fact raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.
- C35A03E, & R: These tests assume that attribute 'MANTISSA returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.
- C37213H: The subtype declaration of SCONS in line 100 is wrongly expected to raise an exception when elaborated.
- C37213J: The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.

WITHDRAWN TESTS

- C37215C, Various discriminant constraints are wrongly expected to be incompatible with type CONS.
E, G, H:
- C38102C: The fixed-point conversion on line 23 wrongly raises CONSTRAINT_ERROR.
C41402A: 'STORAGE_SIZE' is wrongly applied to an object of an access type.
- C45332A: The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOW is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOW may still be TRUE.
- C45614C: REPORT.IDENT_INT has an argument of the wrong type (LONG_INTEGER).
- A74106C, A bound specified in a fixed-point subtype declaration lies outside of
C85018B, that calculated for the base type, raising CONSTRAINT_ERROR. Errors
C87B04B, of this sort occur in lines 37 & 59, 142 & 143, 16 & 48, and 252 & 253
CC1311B: of the four tests, respectively (and possibly elsewhere).
- BC3105A: Lines 159..168 are wrongly expected to be illegal; they are legal.
- AD1A01A: The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.
- CE2401H: The record aggregates in lines 105 & 117 contain the wrong values.
- CE3208A: This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.